Advanced Rexx Workshop

How To Write Self-Healing Rexx Programs

or:

#@%$&!!!

Who can be calling at THIS hour?

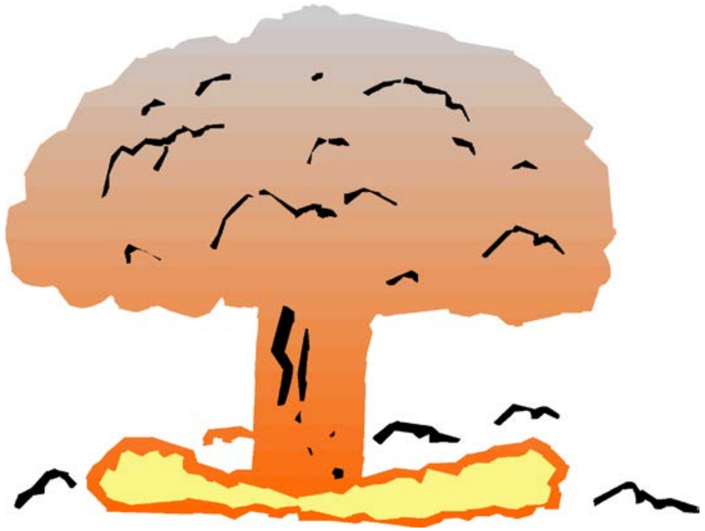
Chip Davis chip@aresti.com

Aresti Systems, LLC

The Problem

A Rexx program with

- Syntax errors or sloppy code
- File locked or not found
- Command failure
- Over-precise arithmetic
- Operator interruption
- Or anything else that can blow up...

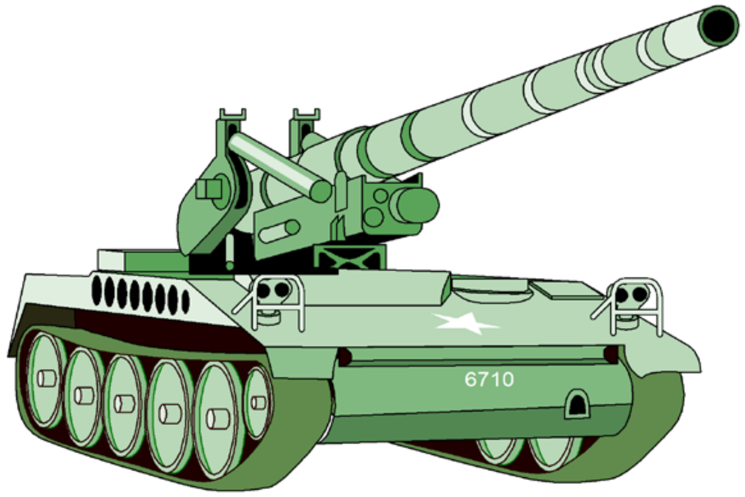


...*will* blow up, usually in the middle of the night!

Your Choice

- Example of normal Rexx error message
- Demo SHealB.cmd without an error trap
- Demo SHealT.cmd with an error trap

VS.

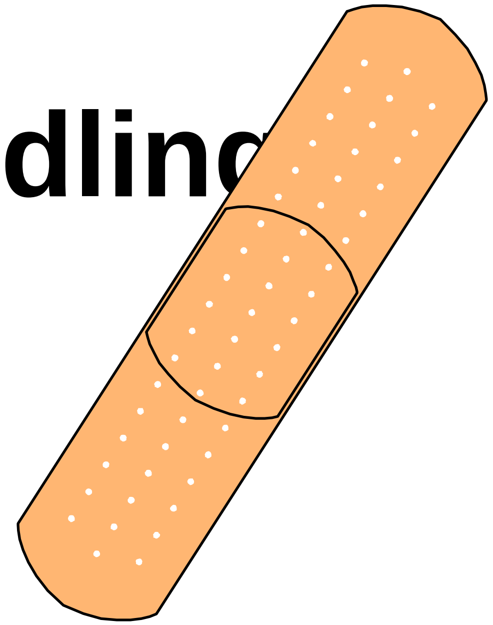


Normal Rexx Error Handling

```
> shealb
Enter full filespec:
test.exe
disk = te
     6 *-* Say "path =" SubStr(fs, 3, lastslash-2)
REX0093E: Error 93 running G:\SHealB.cmd line 6:  Incorrect call to
routine
REX0451E: Error 93.923:  Incorrect length argument specified; found
"-2"
```

- ►Statement listed as if **`TRACE RESULTS`** were set on failing instruction
- ►Two error messages reported
  - ►Major error code = integer — basic error message
  - ►Minor error code = fraction — elaboration - most have values substituted into message
- ►Execution stops, interpreter exits to operating system

Trapped Error Handling

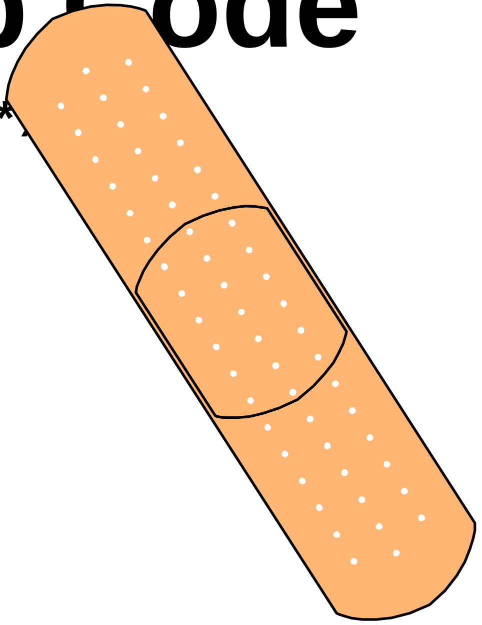


```
> shealt
Enter full filespec:
test.exec
disk = te
```

```
>>> Are you sure that was the complete file specification?
>>> It should have the format: d:\dir\...\dir\file.ext
>>> Please try again.
```

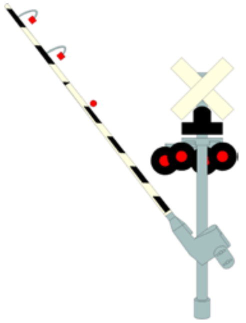
- ►Error is intercepted, allowing more flexible handling of condition
- ►Handle only the errors you wish
  - ►Enable trap for only certain portions of the program - let Rexx handle the rest
  - ►All information necessary to simulate all Rexx major error messages is available
- ►Execution continues with choice of
  - ►Graceful exit following recovery procedures
  - ►Return to resume execution at point of error

Example of Trap Code



```
  /* Signal On Syntax trap  - Chip 26Feb03 */
  Signal On Syntax
  Say "Enter full filespec:"
  Parse Pull fs
  lastslash = LastPos('\', fs)
  Say "disk =" Left(fs, 2)
  Say "path =" SubStr(fs, 3, lastslash-2)
  Say "file =" SubStr(fs, lastslash+1)
  Exit 0
Syntax:
  Say ""
  Say ">>> Are you sure that was the complete file specification?"
  Say ">>> It should have the format: d:\dir\...\dir\file.ext"
  Say ">>> Please try again."
  Exit 99
```

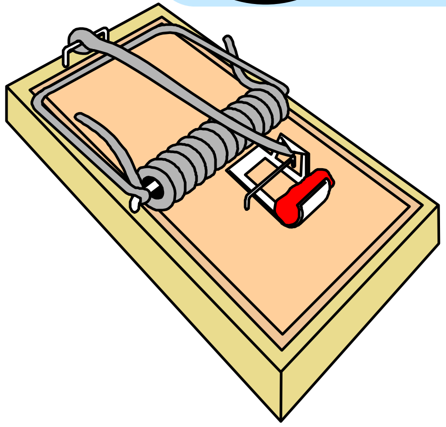
SIGNAL & CALL



```
>> | SIGNAL | | OFF   condition                   |
      CALL      ON    condition |                 |
                                   NAME  routine |
```

condition:

	FAILURE	RC < 0 from a command
	ERROR	RC > 0 from a command *
	NOTREADY	I/O error
*	HALT	external interrupt
SIGNAL only	NOVALUE	referenced an uninitialized variable
	LOSTDIGITS	tried to use a number longer than DIGITS()
*	SYNTAX	anything else that's bad...



* if FAILURE also trapped, otherwise: RC \= 0 from a command

* causes termination even if not trapped

Special Variables: RC & SIGL

- ☒**`RC`**
  - ☒on **`ERROR`** or **`FAILURE`** traps, contains
    - ☒numeric return code from last command issued
  - ☒on **`SYNTAX`** trap, contains
    - ☒number of error message that Rexx would have issued

- ▪**`SIGL`**
  - ►contains line number that transferred execution to here
  - ►use with **`SOURCELINE()`** to display failing instruction
  - ►not just for error trapping...

SourceLine( )

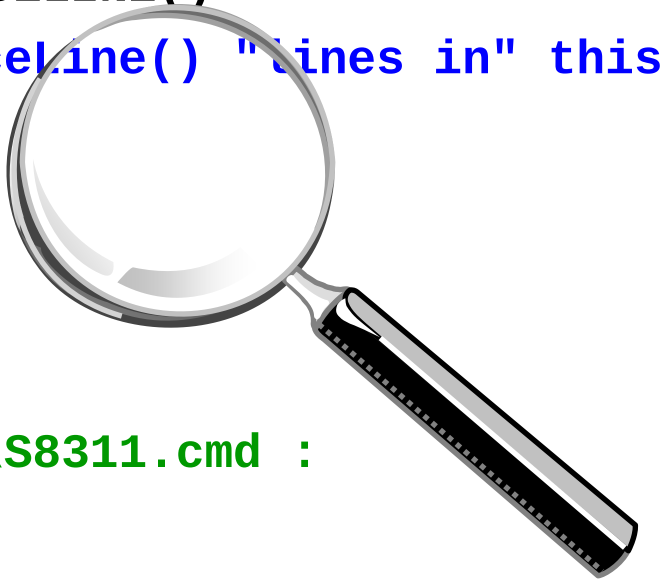
```
>>   SOURCELINE(                    )   ><
                 |  linenumber |
```

linenumber number of source program line to be returned as a string

- Returns number of lines in source program, if omitted
- Returns null, if *linenumber* > `SOURCELINE()`

```
Say "Error on line" sigl "of" SourceLine() "Lines in" thisprog ":"
Say sigl-2 SourceLine(sigl-2)
Say sigl-1 SourceLine(sigl-1)
Say sigl">" SourceLine(sigl)
Say sigl+1 SourceLine(sigl+1)
```

```
Error on line 31 of 54 lines in L:\S8311.cmd :
29   jul = jul + dd
30   day = (jul + offset) // 7
31>  today = Left(Date('S')4) || Date('D')
32   If today < (yyyy || jul) Then verb = 'will be a'
```



ErrorText( )

```
>>    ERRORTEXT(    msgnum    )    ><
```

msgnum number (0-99) of error message to be returned as a string

- ►Returns null, if number not assigned an error message
- ►Returns major error message text only

```
Say ">" thisprog "line" sigl "raised Rexx Error" rc":"
ErrorText(rc)
Say ">" SourceLine(sigl)
```

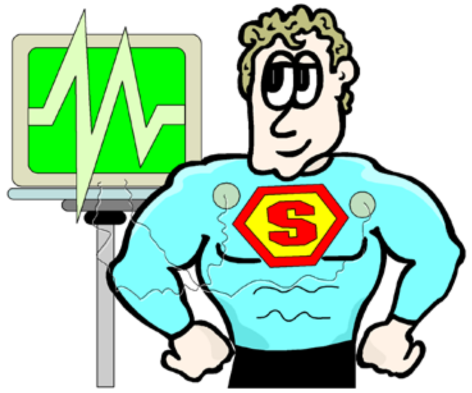
```
> L:\SHealU.cmd [7] raised Rexx Error 40: Incorrect call to routine
>   Say "path =" SubStr(fs, 9.3, lastslash-2)
```

(versus)

```
     8 *-* ft = SubStr(fn,9.3)
REX0040E: Error 40 running L:\SHealU.cmd line 7:  Incorrect call to
routine
```

Condition( )

```
                   Instruction
>>   CONDITION(   |                   ) |        ><
                   Status
                   Condition
                   Description
```



`Instruction`	How we got here: either CALL or SIGNAL
`Status`	State of trap: ON, OFF, or DELAY
`Condition`	Name of trapped condition
`Description`	Trap-specific information, based on `Condition:`

`ERROR`	command string that returned error or failure
`FAILURE`	command string that returned failure
`HALT`	null string
`LOSTDIGITS`	number longer than NUMERIC DIGITS
`NOVALUE`	derived name of uninitialized variable
`NOTREADY`	stream name
`SYNTAX`	null string

Parse Source

```
>>-- PARSE |         | SOURCE    template        ><
               UPPER
```

Source string contains at least 3 tokens:

1 - Where am I?	TSO OS/2 AIX UNIX ...
2 - How was I started?	COMMAND SUBROUTINE FUNCTION
3 - What is my name?	*filename path/filename member ...*
4+ Anything else?	*platform dependent information ...*

```
Parse Source srcstr
Say ">" srcstr

> TSO COMMAND SHealV SYSEXEC CHIP.REXX.EXEC ? TSO ISPF ?

Parse Source srcstr
Say ">" srcstr

> OS/2 FUNCTION L:\SHealV.cmd
```

Trapping HALT

```
/* Call On Halt trap  - Chip 26Feb03 */
  Call On Halt Name Ten20
  Parse Arg n .
  Numeric Digits (Length(n)-1)*2
  sum = 0
  Do i = 1 To n
    sum = sum + i
  End i
  Say "Summation of i (j=1,"n") =" sum
  Exit 0
Ten20:
  Say "i="i "sum=" sum
  Say "To continue, press -Enter-.  Any other key to cancel."
  Parse Pull reply .
  If reply = '' Then Return
  Exit 0
```

Trapping NOTREADY

```
/* Signal On NotReady trap  - Chip 26Feb03 */
  file = "M8DATA"
  If Stream(file, 'Command', 'Open Read') \= 'READY:' Then Exit 28

  Signal On NotReady Name EOF
  Call ReadFile
  Signal Off NotReady          -- Return here at EOF
  Do j = 1 To ans.0            -- Prove that it worked
    Say "Answer" j":" ans.j
  End j
  Exit 0

ReadFile:                      -- Read records until EOF
  Do i = 1
    ans.i = LineIn(file)          -- Only place NOTREADY can be raised!
  End i

EOF:                           -- Close file and return to main routine
  ans.0 = i - 1
  Call Stream file, 'Command', 'Close'
  Return 0
```

RexxTry

```
/* RexxTry - Dynamically execute Rexx intructions */
  Trace Off
  Signal On Syntax Name !Oops!
  Parse Arg rxinst
  If rxinst \= '' Then Return ?Try?()
  Say "" ; Say "Enter any Rexx instruction or 'exit' to quit:"
  Say ""
  Do Forever
    Parse Pull rxinst
    Call ?Try?
    Say "---"; Say ""
  End  /* Forever */
?Try?:
  Interpret rxinst
  Return 0
!Oops!:
  Trace Off
  Say "" ; Say "Oops! That caused error" Rc":" ErrorText(Rc)
  Say "Try again."
  Return 0
```

Conclusion

- Costs very little to add robust error trapping to a Rexx program, either at development time or later
- Pays BIG dividends in uninterrupted sleep
- Accurate metric of programmer professionalism
- But...
  - it's a very sharp knife -- don't mis-handle it
  - "Just because Rexx lets you get away with it, that doesn't mean it's a smart thing to do."